A Crash Course on Machine Learning and Tensorflow

What is Machine Learning?

- Traditional programming (what we've done so far) requires you to specify exactly
- how to complete a task, line by line.

Machine learning refers to the programming techniques that use statistics to allow the program to learn and improve its tasks.

• We do this by building a model with several adjustable parameters that can attempt to accomplish a task.

• The model learns by repeatedly testing the task, then adjusting the parameters slightly to optimize the result approximation.

- Over time, the model's approximation will get closer and closer to the actual value.

Types of ML Tasks

- Machine learning problems can be broken down into a few categories.
- **Supervised Learning:** Tasks where we have a metric for success (labeled data, way of quantifying the accuracy of the result.)

• **Classification:** The model must classify some input data - Filtering spam

- Determining what an image is a picture of

• **Reinforcement Learning:** The model interacts with a dynamic environment by observing the current state of the environment and rewards for good performance.

- Learning to play arcade games.
- **Unsupervised Learning:** Problems where the model has no guidance and must infer patterns and features in the data independently.

How Does ML Work?

- The statistical techniques to perform machine learning tasks have been around for decades, but only recently has the computing power necessary to make it practical become available.
- How does a model "learn?"
• First, data is fed into the model. The data fed into the model is called the training data. The parameters inside the model take that data and return an answer. • The model's guess is then compared with the expected outcome.
- Was it right or was it wrong?
- Then, the parameters are optimized in the direction of the expected outcome.
• When the model has finished learning, it can attempt to predict the outcome of data not in the original training set. This data is called the test data.
• This process is repeated with new data until the model has optimized its parameters to be as likely as possible to guess the right answer.
- The benefits of machine learning are in the speed at which computers can test possible outcomes.
• For example, when you're playing chess, you might only consider moves that "make sense", or that clearly advances you in the direction of your goal. Then, you don't have to consider every possible move.
• A machine learning algorithm doesn't inherently know which moves are "better" than others. Instead, it tries every possible move and sees which moves to end up in a win and which moves to end up in a loss.
- A machine can test these outcomes much more quickly than you or I could!

What's in a model?

- A machine learning model can be as simple as a single variable that changes over time or as complex as a whole network of thousands of neurons estimating complex processes.
• The more parameters a model has, the more nuanced the task it can learn becomes.
- Tuning your machine learning models is an amateur of balancing complexity, processing
time, and model structure to be the best at learning its task.

Intro to Tensorflow

- Tensorflow is a library developed by Google to facilitate the creation and training of machine learning models and neural networks.
- A basic machine learning algorithm in tensorflow needs 3 things: • a loss function -- a way of quantifying how much error there is.
• an optimizer -- A way of "stepping" up or down to reduce the error
• a metric -- how accurate is our algorithm? (e.g. what percentage of images are classified correctly?)

Scalars, Vectors, and Tensors.

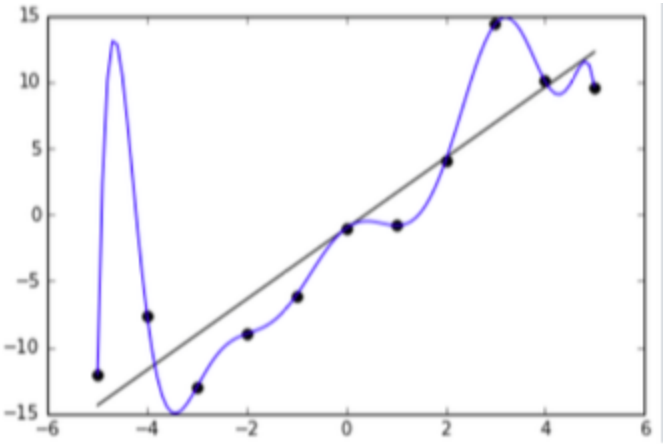
- A scalar is a mathematical object that has magnitude (length), but not direction. • Integer constants and coefficients are examples of scalars.
• 5, 100, -1, 3.5
- A vector is a mathematical object that has magnitude and direction.
• Velocity is a vector because it has magnitude (speed) and direction.
• Vectors are often represented by arrays of numbers that represent their ending position.
- For example, a vector that represents motion from the point (0, 0) to the point (2, 3) is represented as [2, 3]
- A vector that represents motion from the point (1, -5) to the point (2, 5) is represented as [1, 10]
A tensor is the mathematical representation of a physical entity that has magnitude and multiple directions.
• In code, tensors are often represented as an array of vectors, also known as a Matrix.

Image Processing

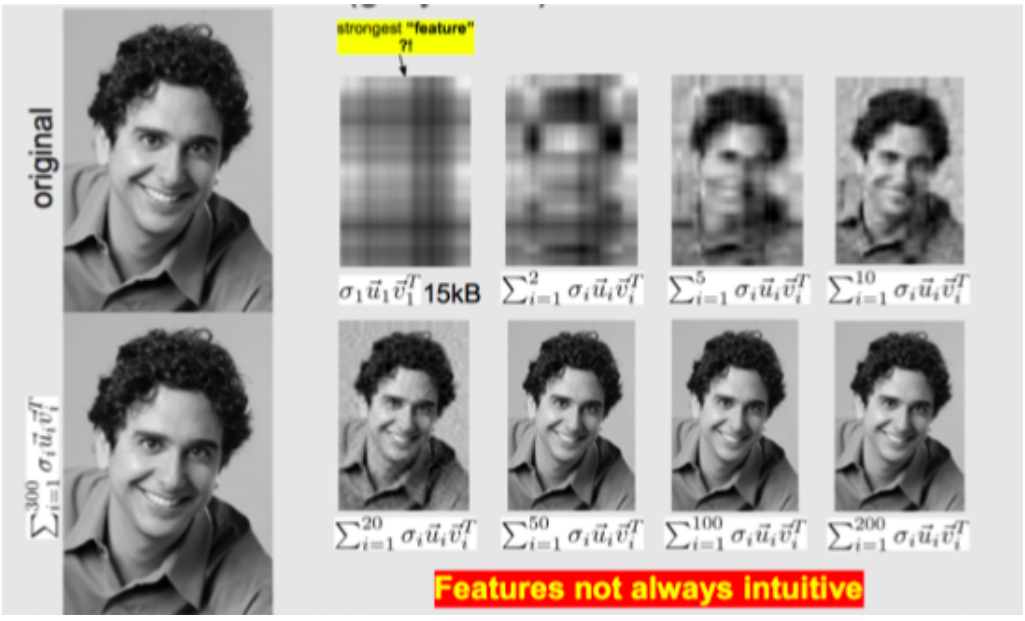
- In the machine learning algorithms we'll look at, images are converted into arrays of pixels, where each pixel is represented by its RGB color.
• For example, 1 black pixel is represented as [0, 0, 0]
- Modern images contain massive amounts of data.
• A 1080p image contains over 2 million individual pixels.
• A 4K image contains over 8.8 million pixels.
• Because of this, we will need to compress, or reduce the number of pixels, in our

images before we can feed them into our algorithm. Otherwise, we risk using too much computational power (having the algorithm take a very long time to run) or, worse, **overfitting** the model.

- A model that has been overfitted will perform very well on the training data, but poorly on any testing data.
  - **Overfitting** is when a model contains more parameters than can be justified by the data.
- For a line of best fit, **overfitting** occurs when we fit a polynomial function (blue) where a linear one will do (black). See the picture to the right.



- Image compression in practice requires lots of linear algebra, specifically a process known as single variable decomposition.
• Essentially, we will deconstruct our image into a sum of rank 1 matrices. These 3 matrices each represent some "feature" in the data.
- We will only use the first few "features" and ignore the rest. An example of this process can be seen below.
- The image here shows the compression of an image of Prof. Maharbiz, an EE professor at UC Berkeley.
• The full image is composed of about 300 "features". However, we see that by the time we include 10 features (upper right images), the image already begins to look recognizable.
- Thus, we only need to use 10 of the 300 features, compressing our image by a significant amount and saving valuable space and computational power.



Application: Linear Regression (Line of Best Fit)

- In order to use tensorflow, we have to **import** the package in our code. # import tensorflow and any necessary dependencies import tensorflow as tf
- import matplotlib.pyplot as plt

The first thing we'll do is set tensorflow **placeholders**.

• Placeholders are like **declarations** of the data type. They tell the program the type of data we'll be feeding into the machine learning algorithm to train it, input_data, as well as the "answers" to the input, output_data.

- For linear regression: The input data is a set of x values, and the "answers" are the corresponding y values.
- For image classification, the input data is a set of images of the type we want to classify (traffic signs, handwritten numbers, etc.), and the correct names for those images are the "answers".

```
# Specify the type and shape of input and output data.
input_data = tf.placeholder(dtype=tf.float32, shape=None)
output_data = tf.placeholder(dtype=tf.float32, shape=None)
```

- The next step is to set tensorflow variables.

• These variables are what we want our machine learning algorithm to solve for.

- Not every program in tensor flow will require **variables**.

• For linear regression, the variables we need to solve for are the slope and y-The intercept of the line of best fit.

```
# Identify parameters to be solved for
slope = tf.Variable(0.5, dtype=tf.float32)
intercept = tf.Variable(0.1, dtype=tf.float32)
```

- Next, we'll define how the placeholders are related to the parameters (the function used to relate the two).

```
# Create the operation used to calculate the guess
model_operation = slope * input_data + intercept
```

- Now, we'll define the **loss** function.

• Recall that in linear regression, we use the **squared error** to determine the loss. This metric is the sum of the squares of the discrepancies between the actual y- value and the y-value predicted using the line of best fit for each data point in the set of x values.

```
# Calculate the error of the algorithm's guess
error = model_operation - output_data
squared_error = tf.square(error)
loss = tf.reduce_mean(squared_error)
```

- Finally, we'll set the **learning rate**, which is kind of like the step size for the graph. The learning rate is an example of a **hyperparameter.**
• We'll also initialize all the global variables.

```
# Set the learning rate hyperparameter and bind the training variable.
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.005) train =
optimizer.minimize(loss)
    # Initialize all our variables.
    init = tf.global_variables_initializer()
```

- To use the algorithm we just created, we need data. In a real-world example, you'd likely need to load an external data set into your python file by specifying the correct **path** in your directory.
• For our toy example, we'll create a small dataset within the file to use as an example.

```
    # Create an arbitrary data set to bind to input_data and
    output_data in the feed_dict.
    x_values = [0, 1, 2, 3, 4]
    y_values = [1, 3, 5, 7, 9]
```

- Finally, we'll run our algorithm 2000 times and pick the best line created out of all of those. (Try changing this number and seeing how the results differ!

```
    # Run the simulation 2000 times using tf.Session()
    with tf.Session() as sess:
        sess.run(init)
        for i in range(2000):
            sess.run(train, feed_dict={input_data:x_values,
    output_data:y_values})
            if i % 100 == 0:
                print(sess.run([slope, intercept]))
        print(sess.run(loss, feed_dict={input_data:x_values,
    output_data:y_values}))
```